

Exhibit 801

(Part 1 of 2)

```

1442 init window(screen,node,output,title,row,col,out_clr,out_fill,out_pat,pane_clr)
1443 register SCREEN *screen;
1444 register MAPNODE *node;
1445 CONNECTOR *output;
1446 register short row,col;
1447 register char *title;
1448 register char out_clr,out_fill,out_pat,pane_clr;
1449 {
1450     register char *msg;
1451     int result = NO;
1452     if (node->style == 's' && (screen->colors < 7 || !screen->bit_map))
1453         node->style = 'S';
1454     if (node->outline)
1455         out_line(node);
1456     if (node->palette)
1457         node->left = node->palette;
1458     if (node->resize_box || node->Vscroll)
1459         node->right += VCHAR WD;
1460     if (node->corner && !node->palette)
1461         node->left += VCHAR WD;
1462     if (node->menu || node->general_use)
1463         node->bottom = VCHAR HT * 2;
1464     else if (node->Hscroll)
1465         node->bottom = VCHAR HT;
1466     align window(screen,node);
1467     msg = Newmsg(3000,"init",
1468         "pos=#2s; size=#2s; outl=#5b; pane=#2b; marg=#4s; scrn=#4s; outp=#C; \
1469         self=#C; map=#C#2s; name=#S; rply",
1470         node->row,node->col,node->height,node->width,
1471         out_clr,node->outline,out_fill,out_pat,node->style,pane_clr,node->pane,
1472         node->top,node->bottom,node->left,node->right,0,0,screen->height,
1473         screen->width,output,&node->window,&node->picture,row,col,node->name);
1474     init frame(msg,node,title,out_clr);
1475     msg = Call(DIRECT,node->window.pid,msg,0,0);
1476     result = strcmp(msg,"failed");
1477     Free(msg);
1478     return(result);
1479 }
1480
1481

```

119

5,502,839

120

```

1482 out_line(node)
1483 register MAPNODE *node;
1484 {
1485     node->outer = node->outline + node->pane + (node->outline && node->pane) *
1486                 (node->height/100 + node->width/100 + 2);
1487     if (node->tight)
1488     {
1489         node->top = node->bottom = node->outer;
1490         node->left = node->right = node->outer + node->width/200;
1491     }
1492     else
1493     {
1494         node->top = VCHAR_HT;
1495         node->bottom = node->outer;
1496         node->left = node->right = VCHAR_WD;
1497     }
1498     if (node->style == 's')
1499     {
1500         node->bottom += 5;
1501         node->right += 5;
1502     }
1503 }
1504

```

121

5,502,839

122

```

1505 init frame(msg,node,title,out_clr)
1506 register MAPNODE *node;
1507 register char *msg, *title, out_clr;
1508 {
1509     char *n, *frame bar();
1510     register char scroll_clr=(4*16)+YELLOW, title_clr = WHITE;
1511     register P E HDR *hdr;
1512     static short up arrow[] = {7,0,0,6,7,12,7,9,10,9,10,3,7,3,7,0};
1513     static short down arrow[] = {3,0,3,3,0,3,0,9,3,9,3,12,10,6,3,0};
1514     static short left arrow[] = {6,0,0,7,3,7,3,10,9,10,9,7,12,7,6,0};
1515     static short right arrow[] = {3,0,3,3,0,3,6,10,12,3,9,3,9,0,3,0};
1516     static long resize_symbol[] = {0x000007f80,0x7f807f80,0x7f807f80,0x7f807f80,
1517     0x7f807fc4,0x00ec007c,0x003c007c,0x00fc0000,0x00000000};
1518
1519     if (node->title)
1520     {
1521         n = frame bar(msg,"top",400,'T',0,0,node->title,1000,0,out_clr,0,NO);
1522         draw filled rect(&n,0,0,node->title,1000,NULL,0,0,out_clr,0,0,0,"a");
1523         draw rect(&n,5,10,10,10,"CLOSE!" title_clr 'S',1,"S");
1524         draw text(&n,0,3*VCHAR_WD,title,"NAME",title_clr,0,NULL,NULL);
1525         if (!node->nonmod)
1526         {
1527             hdr = (P E HDR *) start macro(&n,0,1000,
1528             VCHAR HT-2,2*VCHAR_WD,'N',"FILL!",0,0,"Sa");
1529             draw rect(&n,3,0,VCHAR HT-7,2*VCHAR_WD-4,NULL,title_clr,'S',1,NULL);
1530             draw filled rect(&n,6,3,VCHAR HT-14-2*VCHAR_WD-10,
1531             NULL,0,0,title_clr,0,0,0,0,NULL);
1532             end_macro(&n,hdr);
1533         }
1534         draw_end(&n);
1535     }
1536     if (node->Vscroll)
1537     {
1538         n = frame bar(msg,"right",400,'V',node->pane-1,node->pane-1,790,
1539         node->right-node->pane-node->outline+2,out_clr,BLACK,1,NO);
1540         draw rect(&n,node->pane,node->pane,VCHAR HT-4,
1541         node->right-(node->pane)-(node->outline),
1542         "SCROLL!" scroll_clr,'S',1,"Sb");
1543         draw poly(&n,875,node->pane+1,
1544         8,up arrow,"UP!",scroll_clr,0,0,'S',0,1,"Sa");
1545         draw poly(&n,980,node->pane+1,
1546         8,down arrow,"DOWN!",scroll_clr,0,0,'S',0,1,"Sa");
1547         draw_end(&n);
1548     }
1549 }

```

123

5,502,839

124

```

1550
1551 if (node->Hscroll)
1552 {
1553     n = frame_bar(msg, "bot ", 400, 'H', node->pane-1, 0,
1554                 node->bottom-(node->pane)-(node->outline)+2,
1555                 910, out_clr, BLACK, 1, NO);
1556     draw_rect(&n, node->pane, node->pane,
1557             node->bottom-node->pane-node->outline,
1558             2*VCHAR WD-2, "SCROLL!", scroll_clr, 'S', 1, "sb");
1559     draw_poly(&n, node->pane, 955,
1560             8, left_arrow, "LEFT!", scroll_clr, 0, 0, 'S', 0, 1, "Sa");
1561     draw_poly(&n, node->pane, 990,
1562             8, right_arrow, "RIGHT!", scroll_clr, 0, 0, 'S', 0, 1, "Sa");
1563     draw_end(&n);
1564 }
1565 if (node->menu)
1566     frame_bar(msg, "bot ", 200, 'M', node->pane-1, 0,
1567             node->bottom-(node->pane)-(node->outline)+2,
1568             1000, out_clr, BLACK, 1, YES);
1569 if (node->general_use)
1570     frame_bar(msg, "bot ", 200, 'G', node->pane-1, 0,
1571             node->bottom-(node->pane)-(node->outline)+2,
1572             1000, out_clr, BLACK, 1, YES);
1573 if (node->palette)
1574     frame_bar(msg, "left", 200, 'P', 0, node->pane, 10000,
1575             node->left-(node->pane)-(node->outline)-1, out_clr, BLACK, 1, YES);
1576 if (node->resize_box)
1577 {
1578     n = frame_bar(msg, "rbox", 200, NULL, 0, 0, 0, 0, scroll_clr, BLACK, 1, NO);
1579     draw_symbol(&n, 0, 0, 16, 16, resize_symbol, "RESIZE!", scroll_clr, 0, "S");
1580     draw_end(&n);
1581 }
1582 if (node->corner)
1583     frame_bar(msg, "lbox", 200, NULL, 0, 0, 0, 0, out_clr, BLACK, 1, YES);
1584 }

```

125

5,502,839

126

```

1585 char *frame bar(msg, keyw, size, type, row, col, height, width, color, fill, thick, end)
1586 register char *msg, *keyw;
1587 char type, color, fill, end;
1588 register short row, col, height, width, size, thick;
1589 {
1590     char *n;
1591     n = Append triple(msg, keyw, size, NULL);
1592     *n++ = type;
1593     draw filled rect(&n, row, col, height, width,
1594                     NULL, color, 0, fill, 0, 'S', 0, thick, "a");
1595     if (end)
1596         draw end(&n);
1597     return(n);
1598 }
1599
1600 Set user(name, buf, size)
1601 register NAME *name;
1602 register long buf, size;
1603 {
1604     register char *p;
1605     if (p = Find_triple(buf, "name", size, NULL, 2, NULL))
1606     {
1607         strcpy(name->user, p);
1608         Note("signed on", p);
1609         Put(ALL, "HI", Newmsg(128, "U", "name=#S", p));
1610     }
1611 }
1612
1613
1614

```

127

5,502,839

128

```

1615 Change(screen,map,msg)
1616 SCREEN      *screen;
1617 LIST        *map;
1618 MESSAGE     *msg;
1619 {
1620     register CONNECTOR *window, *owner = NULL;
1621     register short *p;
1622     register MAPNODE *node;
1623     if (window = (CONNECTOR*)Find_triple(msg->buf,"conn",msg->size,NULL,8,NULL))
1624     {
1625         for (node = map->first; node && node->window.pid != window->pid
1626             && node->terminal.pid != window->pid; node = node->nxt) ;
1627         if (node)
1628         {
1629             if (p = (short*) Find_triple(msg->buf,"size",msg->size,none,4,NULL))
1630             {
1631                 resize(screen,node,*p,*(p+1));
1632                 if (Find_triple(msg->buf,"actv",msg->size,NULL,0,NULL)
1633                     && !node->never)
1634                     map->active = node;
1635                 if (owner =
1636                     (CONNECTOR*) Find_triple(msg->buf,"ownr",msg->size,NULL,0,NULL))
1637                     if ((long)owner == 1)
1638                         owner = &msg->sender;
1639                 if (owner)
1640                 {
1641                     node->owner = *owner;
1642                     if (node->terminal.pid)
1643                     {
1644                         Forward(DIRECT,node->terminal.pid,msg->buf);
1645                         msg->buf = NULL;
1646                     }
1647                 }
1648             }
1649             clip_window(map->last);
1650         }
1651     }
1652 }

```

129

5,502,839

130

```

1653 highlight(node, map)
1654 register MAPNODE      *node;
1655 register LIST         *map;
1656 {
1657     if (node && node != map->last_active)
1658     {
1659         if (!node->metaphor)
1660         {
1661             Put(LOCAL, "Window"
1662                 Newmsg(64, "highlight", "bar=#b; tag=#S", 'T', "CLOSE!"));
1663             if (node->window.pid && node->title)
1664                 Put(DIRECT, node->window.pid,
1665                     Newmsg(128, "highlight", "off; bar=#b; tag=#S", 'T', "CLOSE!"));
1666             if (node->window.pid)
1667                 Put(DIRECT, node->window.pid, Newmsg(32, "keys?", NULL));
1668             map->last_active = node;
1669         }
1670     }
1671 }
1672
1673 move mark(row, col, picture)
1674 register short      row, col;
1675 register CONNECTOR  *picture;
1676 {
1677     Put(DIRECT, picture->pid, Newmsg(32, "mark", "at=#2s", row, col));
1678 }
1679
1680

```

131

5,502,839

132


```

1681 clip window(node)
1682 register MAPNODE *node;
1683 {
1684     register MAPNODE *temp;
1685     register short prio = 127, count, *count_addr, *n;
1686     char *m;
1687     for ( ; node; node = node->pre)
1688     {
1689         m = Newmsg(1000,"cut", "inHI=#s#s#A",prio--,0,950,NULL);
1690         count_addr = (short *) (Find_triple(m,"inHI",0,NULL,0,NULL) + 2);
1691         n = count_addr + 1;
1692         count = 0;
1693         for (temp = node->pre; temp; temp = temp->pre)
1694         {
1695             *n++ = temp->row;
1696             *n++ = temp->col;
1697             *n++ = temp->out_ht;
1698             *n++ = temp->out_wd;
1699             count++;
1700         }
1701         *count_addr = count;
1702         Put(DIRECT,node->window.pid,m);
1703     }
1704 }
1705
1706 MAPNODE *find window(map,window,row,col)
1707 register LIST *map;
1708 register WINDOW *window;
1709 register short row, col;
1710 {
1711     register MAPNODE *node;
1712     for (node = map->first; node; node = node->nxt)
1713     {
1714         query window(window,node->window,row,col);
1715         if (window->area != 'N')
1716             break;
1717     }
1718     window->previous = window->node;
1719     return(window->node = node);
1720 }
1721
1722
1723

```

133

5,502,839

134

```

1724 query window(window,conn,row,col)
1725 register WINDOW *window;
1726 CONNECTOR conn;
1727 register short row, col;
1728 {
1729     register char *p, *reply;
1730     if (window->hdr)
1731         Free(window->hdr);
1732     window->hdr = NULL;
1733     window->elem_row = window->elem_col = -1;
1734     reply = Call(DIRECT,conn.pid,Newmsg(64,"w","inHI=#2s",row,col),0,0);
1735     p = FindTriple(reply,"inHI",0,none,1,NULL);
1736     p += 2 * sizeof(short);
1737     window->area = *p++;
1738     window->bar = *p++;
1739     window->row = *{(short *) p}++;
1740     window->col = *{(short *) p}++;
1741     LongAlign(p);
1742     if (*(short*)p)
1743     {
1744         window->hdr = (P E HDR *) Alloc(*(short*)p,YES);
1745         memcpy(window->hdr,p,*(short*)p);
1746     }
1747     Free(reply);
1748 }
1749
1750 MAPNODE *new node(map,name)
1751 register LIST *map;
1752 register char *name;
1753 {
1754     register MAPNODE *node = NULL;
1755     register short i;
1756     for (i = POOL_SIZE, node = map->pool; node->pool && i; ++node, --i) ;
1757     if (!i)
1758         node = (MAPNODE *) Alloc(sizeof(MAPNODE),YES);
1759     memset(node,0,sizeof(MAPNODE));
1760     node->pool = i;
1761     strcpy(node->name,name);
1762     return(node);
1763 }
1764
1765
1766

```

135

5,502,839

136

```

1767 free node(node)
1768 register MAPNODE *node;
1769 {
1770     if (node->pool)
1771         node->pool = NULL;
1772     else
1773         Free(node);
1774 }
1775
1776 map_after(node, pred, map)
1777 register MAPNODE *node, *pred;
1778 register LIST *map;
1779 {
1780     if (pred)
1781     {
1782         node->nxt = pred->nxt;
1783         node->pre = pred;
1784         if (pred->nxt)
1785             (pred->nxt)->pre = node;
1786         pred->nxt = node;
1787     }
1788     else
1789     {
1790         if (node->nxt = map->first)
1791             (map->first)->pre = node;
1792         node->pre = NULL;
1793     }
1794     if (!node->pre)
1795         map->first = node;
1796     if (!node->nxt)
1797         map->last = node;
1798     ++map->count;
1799 }
1800
1801 unmap(node, map)
1802 register MAPNODE *node;
1803 register LIST *map;
1804 {
1805     if (node->pre)
1806         (node->pre)->nxt = node->nxt;
1807     else
1808         map->first = node->nxt;
1809     if (node->nxt)
1810         (node->nxt)->pre = node->pre;
1811     else
1812         map->last = node->pre;
1813     --map->count;
1814 }
1815

```

137

5,502,839

138

```

1816 remap(window,node,new_picture,map,sel)
1817 register CONNECTOR -*window, *new_picture;
1818 register MAPNODE *node;
1819 register SELECTION *sel;
1820 LIST *map;
1821 (
1822     if (window)
1823         for (node = map->first;
1824             node && window->pid != node->window.pid; node = node->nxt) ;
1825     if (node)
1826     {
1827         end edit(node,'X',0,0,NULL);
1828         if (new_picture && new_picture->pid != node->picture.pid)
1829             if (node == sel->map)
1830             {
1831                 sel->map = NULL;
1832                 sel->pehding = NO;
1833             }
1834         node->picture = *new_picture;
1835     }
1836 )
1837

```

139

5,502,839

140

```

1838 align window(screen,node)
1839 register SCREEN *screen;
1840 register MAPNODE *node;
1841 (
1842     register short temp;
1843     if (screen->char_align)
1844     (
1845         if (node->tight)
1846         (
1847             temp = ((node->row % VCHAR_HT) || node->outer ? VCHAR_HT : 0);
1848             node->row = (node->row / VCHAR_HT) * VCHAR_HT - node->outer + temp;
1849             temp = ((node->col % VCHAR_WD) || node->outer ? VCHAR_WD : 0);
1850             node->col = (node->col / VCHAR_WD) * VCHAR_WD -
1851                 (node->outer + node->width/200) + temp;
1852         )
1853         else
1854         (
1855             node->row = ((node->row + VCHAR_HT-1) / VCHAR_HT) * VCHAR_HT;
1856             node->col = ((node->col + VCHAR_WD-1) / VCHAR_WD) * VCHAR_WD;
1857         )
1858         if (node->row < screen->meta_row)
1859             node->row += (screen->meta_row + VCHAR_HT-1) / VCHAR_HT * VCHAR_HT;
1860         if (node->col < screen->meta_col)
1861             node->col += (screen->meta_col + VCHAR_WD-1) / VCHAR_WD * VCHAR_WD;
1862         if (node->out_ht > screen->meta_ht)
1863             node->height = screen->meta_ht - (node->top + node->bottom);
1864         if (node->out_wd > screen->meta_wd)
1865             node->width = screen->meta_wd - (node->left + node->right);
1866         if (!node->tight)
1867         (
1868             temp = (node->height % VCHAR_HT ? VCHAR_HT : 0);
1869             node->height = (node->height / VCHAR_HT) * VCHAR_HT + temp;
1870             temp = (node->width % VCHAR_WD ? VCHAR_WD : 0);
1871             node->width = (node->width / VCHAR_WD) * VCHAR_WD + temp;
1872         )
1873     )
1874     node->out_ht = node->height + node->top + node->bottom;
1875     node->out_wd = node->width + node->left + node->right;
1876 )
1877
1878

```

141

5,502,839

142

```

1879
1880 Status(msg, size)
1881 register char *msg;
1882 register long size;
1883 {
1884     register char *m;
1885
1886     *(m = Alloc(size, YES)) = NULL;
1887     strcat(m, Find triple(msg, "orig", size, none, 1, NULL));
1888     strcat(m, "-");
1889     strcat(m, Find triple(msg, "stat", size, none, 1, NULL));
1890     strcat(m, "in-");
1891     strcat(m, Find triple(msg, "req ", size, none, 1, NULL));
1892     Note(m, "ERROR");
1893     Free(m);
1894 }
1895
1896 reply_status(req, mid, stat, code)
1897 register char *req, *mid, *stat;
1898 register long *code;
1899 {
1900     register char *type, *msg;
1901
1902     type = "failed";
1903     if (!mid)
1904         type = "status";
1905     else if (*mid == '-')
1906         mid++;
1907     else if (*mid == '+')
1908     {
1909         type = "done";
1910         mid++;
1911     }
1912     msg = Newmsg(strlen(stat)+100, type,
1913 "orig=#S; stat=#S; code=#I", "console", stat, code);
1914     if (mid)
1915     {
1916         Append triple(msg, "req ", strlen(mid)+1, mid);
1917         Reply(req, msg);
1918     }
1919     else
1920         Put(DIRECT, (long) req, msg);
1921 }
1922
1923 info(dialogue, string, window)
1924 CONNECTOR dialogue, window;
1925 register char *string;
1926 {
1927     Put(DIRECT, dialogue, pid, Newmsg(strlen(string)+100, "info",
1928 "text=#S; near=#C; wait=#S", string, &window, 5));
1929 }

```

143

5,502,839

144

```

9      Module      : %M% %I%
10     Date submitted : %E% %U%
11     Author       : Frank Kolnick
12     Origin       : cX
13     Description   : Picture Manager
14
15     *****/
16
17     #ifndef lint
18     static char SrcId[] = "%Z% %M%:%I%";
19     #endif
20     /* Picture manager: global data */
21
22     #include <cX.h> /* cX definitions */
23     #include <HI.h> /* picture, etc. definitions */
24     #include <memory.h>
25     #include <string.h>
26     static long none = 0;
27
28     typedef struct element_node /* links picture elements: */
29     {
30         struct element_node *nxt; /* ->next node */
31         struct element_node *pre; /* ->preceding node */
32         unsigned char changed; /* element has changed */
33         unsigned char marked; /* element is marked */
34         unsigned char deleted; /* no longer in use */
35         unsigned char pool; /* local buffer pool */
36         short length; /* (start of element) */
37         /***** NOTE: 'length' must start on a long-word boundary *****/
38     } ELEMENT;
39
40     typedef struct current_state /* current data: */
41     {
42         char *msg; /* ->current msg. */
43         CONNECTOR sender; /* conn. to msg. sender */
44         long size; /* size of msg. */
45         long appl; /* relevant application */
46         short appl_row, appl_col; /* application 'origin' */
47         CONNECTOR owner; /* conn. to owning proc. */
48         char *mark; /* current mark element */
49         char *old mark; /* copy of previous mark */
50         char *erase mark; /* element to erase mark */
51         unsigned char display mark; /* 'display mark' */
52         unsigned char private; /* 'private picture' */
53         unsigned char check; /* 'check size' */
54         unsigned char debug; /* 'print diagnostics' */
55         char highlight; /* type of highlighting */
56         char name[32]; /* picture's name */
57         char file[64]; /* picture file's name */
58         long status code; /* current status ... */
59         char *status_string; /* ... */
60     } CURRENT;

```

PROGRAM LISTING B

145

5,502,839

146

```

61
62
63 typedef struct view_node
64 {
65     struct view_node *nxt;
66     CONNECTOR owner;
67     short row, col;
68     short height, width;
69 } VIEW;
70
71 typedef struct appl_node
72 {
73     struct appl_node *nxt;
74     long name;
75     CONNECTOR conn;
76     short row, col;
77 } APPL;
78
79 typedef struct anim_node
80 {
81     struct anim_node *nxt;
82     long name;
83     CONNECTOR conn;
84 } ANIM;
85
86 typedef struct affected_area
87 {
88     short r1, c1;
89     short r2, c2;
90     char color;
91     char pattern;
92     short max_height;
93     short max_width;
94     short height, width;
95 } AREA;
96
97 typedef struct lists
98 {
99     ELEMENT *first;
100    ELEMENT *last;
101    ELEMENT *current;
102    VIEW *views;
103    APPL *appls;
104    ANIM *ahims;
105    int changes;
106    int erases;
107    int size;
108    struct
109    {
110        long n;
111        long size;
112        ELEMENT *ptr;
113    } pool;
114 } LIST;
115
116

```

```

/* links viewports: */
/*
    ->next node */
/*
    owner of viewport */
/*
    start of viewport */
/*
    extent */

/* links applications: */
/*
    ->next node */
/*
    name of application */
/*
    conn. to application */
/*
    origin */

/* links animation processes: */
/*
    ->next node */
/*
    name of element */
/*
    conn. to process */

/* area changed by a request: */
/*
    upper left front */
/*
    lower right back */
/*
    background color */
/*
    background pattern */
/*
    max. height */
/*
    max. width */
/*
    current size */

/* list pointers, etc.: */
/*
    ->pict. element list */
/*
    ->end of p.e. list */
/*
    ->last p.e. changed */
/*
    ->viewport list */
/*
    ->applications list */
/*
    ->animation list */
/*
    changes in list */
/*
    erasures in list */
/*
    picture elements */
/*
    element pool descr.: */
/*
    #elements */
/*
    size of elements */
/*
    ->element buffer */

```

147

5,502,839

148


```

117  /* local functions */
118
119  char      *value(), *tag();
120  ELEMENT   *mark_number(), *mark_area(), *mark_elements(), *new_element();
121  P_E_HDR   *first_macro(), *next_macro();
122
123  /* Picture manager: main-line */
124
125  PROCESS(Picture)
126  {
127      CURRENT      cur;
128      AREA          area;
129      LIST          list;
130      register VIEW *view;
131      register ANIM *anim;
132
133      Set event key("Picture mgr.");
134      init PM(&cur, &area, &list);
135      draw_picture(&cur, &area, &list);
136      for (view = list.views; view; view = view->nxt)
137          Put(DIRECT, view->owner.pid, Newmsg(32, "unmap", NULL));
138      for (anim = list.anims; anim; anim = anim->nxt)
139          Put(DIRECT, anim->conn.pid, Newmsg(32, "quit", NULL));
140      Exit();
141  }
142
143  init PM(cur, area, list)
144  register CURRENT *cur;
145  register AREA *area;
146  register LIST *list;
147  {
148      area->color = BLACK;
149      area->pattern = 0;
150      *cur->name = *cur->file = NULL;
151      area->max height = area->max width = 0;
152      list->current = list->first = list->last = NULL;
153      list->views = NULL;
154      list->appls = NULL;
155      list->ahims = NULL;
156      list->size = list->pool.n = 0;
157      cur->debug = cur->check = cur->private = cur->display_mark = NO;
158      cur->mark = cur->old_mark = cur->erase_mark = NULL;
159  }

```

149

5,502,839

150

```

160
161 draw picture(cur,area,list)
162 CURRENT *cur;
163 register AREA *area;
164 register LIST *list;
165 {
166     register char *msg;
167     register short transaction = 0, result = 0, go = YES;
168     register ELEMENT *element;
169     long status[11], list_size = 0, *req = NULL;
170
171     while (go)
172     {
173         cur->msg = msg = Get(0,&cur->sender,&cur->size);
174         if (!transaction)
175         {
176             list->changes = list->erases = area->r2 = area->c2 = 0;
177             area->r1 = area->c1 = 32767;
178             cur->appl = NULL;
179             if (list->appls)
180                 check_appl(cur,list->appls);
181         }
182         if (*msg == '[' && transaction < 10)
183             status[++transaction] = 0;
184         else if (*msg == ']')
185             --transaction;
186         else
187             go = Request(cur,area,list,msg,cur->size,cur->appl);
188         if (!transaction)
189         {
190             if (list->changes)
191                 notify(cur,area,list);
192             for (element = list->first; element; element = element->nxt)
193             {
194                 element->changed = element->marked = NO;
195                 if (element->deleted && !Any msg(NULL))
196                     delete_element(list,element);
197             }
198             if (Find triple(msg,"rply",cur->size,NO,0,NULL) && result >= 0)
199                 reply_status(msg,msg,"completed",result);
200         }
201         free_requests(msg,cur->size,&req,&list_size);
202     }
203 }

```

151

5,502,839

152

```

204
205 check_appl(cur,appl)
206 register CURRENT *cur;
207 register APPL *appl;
208 {
209     for ( ; appl && (appl->conn.pid != cur->sender.pid); appl = appl->nxt);
210     if (appl)
211     {
212         if (!(cur->appl = appl->name))
213             cur->appl = -1;
214         cur->appl_row = appl->row;
215         cur->appl_col = appl->col;
216     }
217 }
218
219 free_requests(msg,size,req,list_size)
220 register char *msg, **req;
221 register long size, *list_size;
222 {
223     register char *temp, *next;
224
225     if (msg)
226     {
227         *(char**)msg = *req;
228         *req = msg;
229         *list_size += size;
230         if (!Any(msg,NULL) || *list_size > 1000)
231             for (temp = *req, *req = NULL, *list_size = 0; temp; temp = next)
232             {
233                 next = *(char**)temp;
234                 Free(temp);
235             }
236     }
237 }
238
239 Request(cur,area,list,msg,size,appl)
240 register CURRENT *cur;
241 register AREA *area;
242 register LIST *list;
243 register long msg, size, appl;
244 {

```

153

5,502,839

154

```

245 register short go = YES;
246
247 if (!strcmp(msg,"write"))
248     Draw(list,msg,size);
249 else if (!strcmp(msg,"edit"))
250     Edit_text(cur,area,list,msg,size,appl);
251 else if (!strcmp(msg,"mark"))
252     Move_mark(cur,area,list);
253 else if (!strcmp(msg,"hit"))
254     Hit(list,msg,size,appl);
255 else if (!strcmp(msg,"move"))
256     Move(area,list,msg,size,appl);
257 else if (!strcmp(msg,"erase"))
258     Erase(area,list,msg,size,appl);
259 else if (!strcmp(msg,"read"))
260     Copy(cur,area,list,msg,size,appl);
261 else if (!strcmp(msg,"replace"))
262     Replace(area,list,msg,size,appl);
263 else if (!strcmp(msg,"change"))
264     Change(area,list,msg,size,appl);
265 else if (!strcmp(msg,"animate"))
266     Animate(cur,list);
267 else if (!strcmp(msg,"alter") || !strcmp(msg,"cancel"))
268     Alter(cur,list);
269 else if (!strcmp(msg,"number"))
270     Query_number(list,msg,size,appl);
271 else if (!strcmp(msg,"mark?"))
272     Query_mark(cur);
273 else if (!strcmp(msg,"save"))
274     Save_picture(cur,list);
275 else if (!strcmp(msg,"set"))
276     Set_mark(cur,area,list);
277 else if (!strcmp(msg,"restore"))
278     Restore_mark(cur,area,list);
279 else if (!strcmp(msg,"bkgd"))
280     Background(area,list,msg,size);
281 else if (!strcmp(msg,"create"))
282     go = New_picture(cur,area,list);
283 else if (!strcmp(msg,"init"))
284     cur->private = go = New_picture(cur,area,list);
285 else if (!strcmp(msg,"open"))
286     go = Old_picture(cur,list);
287 else if (!strcmp(msg,"appl"))
288     Appl(cur,list);
289 else if (!strcmp(msg,"quit"))
290     (
291         if (go = (cur->sender.pid != cur->owner.pid))
292             reply_status(msg,msg,"not authorized",0);
293     )

```

155

5,502,839

156

```

294     else if (!strcmp(msg, "query"))
295         Query(cur, list);
296     else if (!strcmp(msg, "failed"))
297         Status(msg, size);
298     else if (!strcmp(msg, "done") || !strcmp(msg, "status"))
299         ;
300     else if (!Change_attribute(list, msg, size, appl))
301     {
302         if (!strcmp(msg, "view"))
303             Viewport(cur, area, list);
304         else if (!strcmp(msg, "debug"))
305             cur->debug = !cur->debug;
306         else
307             reply_status(msg, "-\ 'unknown\ '", msg, 0);
308     }
309     return(go);
310 }

```

157

5,502,839

158

```

311
312 Change attribute(list,msg,size,appl)
313 register LIST *list;
314 register long msg, size, appl;
315 {
316     static char msgids[] = "select\0blink\0invert\0hide\0highlight\0";
317
318     register char *p;
319     register short new_state, changed, type;
320     register ELEMENT *element;
321     register P_E_HDR *hdr;
322
323     for (p = msgids, type = 0; *p && strcmp(msg,p); p += strlen(p)+1, ++type);
324     if (!*p)
325         return(NO);
326     list->current = element = mark_elements(list,NULL,NULL,msg,size,appl);
327     new_state = !(short)Find_triple(msg,"off ",size,NO,0,NULL);
328     for( ; element; element = element->nxt)
329         if (element->marked)
330         {
331             hdr = (P_E_HDR *) &element->length;
332             switch (type)
333             {
334                 case 0: changed = hdr->attr.selected || new_state;
335                         if (hdr->attr.selected == new_state)
336                             Put(NEXT,"Console",Newmsg(hdr->length+50,
337                                     "write","data=%e#e; type=%c",hdr,NULL,'P'));
338                         break;
339                 case 1: changed = hdr->attr.blink != new_state;
340                         hdr->attr.blink = new_state;
341                         break;
342                 case 2: changed = hdr->attr.invert != new_state;
343                         hdr->attr.invert = new_state;
344                         break;
345                 case 3: changed = hdr->attr.hidden != new_state;
346                         hdr->attr.hidden = new_state;
347                         break;
348                 case 4: changed = hdr->attr.highlight != new_state;
349                         hdr->attr.highlight = new_state;
350
351             }
352             if (element->changed == changed)
353                 list->changes++;
354             element->marked = NO;
355         }
356     return(YES);
}

```

159

5,502,839

160

```

357 Query(cur,list)
358 CURRENT *cur;
359 register LIST *list;
360 {
361     unsigned n_elem = 0, n_views = 0;
362     register unsigned min_r = 65535, min_c = 65535;
363     register unsigned max_r = 0, max_c = 0, pic_ht = 0, pic_wd = 0;
364     register ELEMENT *element;
365     register P E HDR *hdr;
366     register VIEW *view;
367
368     for (element = list->first; element; element = element->nxt)
369     {
370         hdr = (P E HDR *) &element->length;
371         if (hdr->row < min_r) min_r = hdr->row;
372         if (hdr->col < min_c) min_c = hdr->col;
373         if (hdr->row + hdr->height > max_r) max_r = hdr->row + hdr->height;
374         if (hdr->col + hdr->width > max_c) max_c = hdr->col + hdr->width;
375         n_elem++;
376     }
377     if (n_elem)
378     {
379         pic_ht = max_r - min_r;
380         pic_wd = max_c - min_c;
381     }
382     else
383         pic_ht = pic_wd = max_r = max_c = min_r = min_c = 0;
384     for (view = list->views; view; view = view->nxt, n_views++);
385     Reply(cur->msg,
386           Newmsg(256,"status","orig=#s; size=#2s; low=#2s; high=#2s; cnt=#s; \
387                view=#s; name=#s; file=#s","picture"
388                pic_ht,pic_wd,min_r,min_c,max_r,max_c,n_elem,n_views,
389                cur->name,cur->file));
390 }
391
392 Query_number(list,msg,size,appl)
393 register LIST *list;
394 register long msg, size, appl;
395 {
396     register unsigned n = 0;
397     register ELEMENT *element, *temp;
398
399     if (element = mark_elements(list,NULL,NULL,msg,size,appl))
400     {
401         for (temp = list->first; temp != element; temp = temp->nxt, n++);
402         Reply(msg,Newmsg(element->length+32,"number"
403                          "num=#s; elem=#e",n,&element->length));
404     }
405     else
406         reply_status(msg,"-number","too high",0);
407 }
408

```

161

5,502,839

162

```

409 Draw(list,msg,size)
410 register LIST *list;
411 register long msg, size;
412 {
413     register ELEMENT *after;
414     register long *p;
415
416     if (p = (long *) Find_triple(msg,"data",size,NULL,4,NULL))
417     {
418         if (Find_triple(msg,"back",size,NO,0,NULL))
419             after = NULL;
420         else
421             after = list->last;
422         if (!draw_elements(p,*p,list,after))
423             reply_status(msg,"-write","bad length/type/macro",0);
424     }
425     else
426         reply_status(msg,"-write","missing \'data\'",0);
427 }
428
429 draw_elements(p,list,len,list,after)
430 register char *p;
431 register long list len;
432 register LIST *list;
433 register ELEMENT *after;
434 {
435     register ELEMENT *element;
436     register short length, number = 0;
437
438     while ((length = *(short *) p)
439            && (list len -= length) >= 0
440            && strchr("tlreacdsmn",((P_E_HDR*)p)->type))
441     {
442         if (((P_E_HDR*)p)->type == 'm' && !check_macro(p))
443             break;
444         element = new_element(list,length+sizeof(ELEMENT),after);
445         memcpy(&element->length,p,length);
446         if (!((P_E_HDR*)p)->height)
447             define_box(&element->length);
448         number++;
449         p += length;
450         Long_align(p);
451     }
452     list->size += number;
453     list->changes += number;
454     list->current = element;
455     return(length ? NO : YES);
456 }
457

```

163

5,502,839

164


```

458
459 define box(hdr)
460 register P_E_HDR    *hdr;
461 {
462     register char    *val;
463
464     val = value(hdr);
465     if (hdr->type == 't')
466     {
467         hdr->height = VCHAR_HHT;
468         hdr->width = VCHAR_WD * strlen(val+8);
469     }
470     else if ((hdr->type == 'n') || (hdr->type == 'm'))
471     ;
472 }
473
474 check_macro(hdr)
475 register P_E_HDR    *hdr;
476 {
477     register P_E_HDR    *temp, *first;
478     short                len;
479     char                 *p, macro_type;
480
481     for (first = temp = first_macro(hdr, &macro_type, &len, &p); temp;
482         temp = next_macro(&len, &p))
483     {
484         if (macro_type == 'L')
485             temp->attr.hidden = YES;
486         if (!temp->height)
487             define_box(temp);
488     }
489     if (macro_type == 'L')
490         first->attr.hidden = NO;
491     return(p ? YES : NO);
492 }

```

165

5,502,839

166

```

493
494 P E HDR *first macro(hdr, type, len, p)
495 register P E HDR *hdr;
496 register char *type;
497 register short *len;
498 register char **p;
499 {
500     register P_E_HDR *temp;
501     *p = value(hdr);
502     if (type)
503         *type = **p;
504     (*p)++;
505     Long_align(*p);
506     temp = (P_E_HDR *) *p;
507     *len = hdr->length - (*p - (char *) hdr);
508     if (temp->length && temp->length < *len && strchr("tlreacdsnm", temp->type))
509         return(temp);
510     *p = NULL;
511     return(NULL);
512 }
513
514 P E HDR *next macro(len, p)
515 register short *len;
516 register char **p;
517 {
518     register P_E_HDR *temp;
519     if (*p)
520     {
521         temp = (P_E_HDR *) *p;
522         *p += temp->length;
523         Long_align(*p);
524         *len -= (*p - (char *) temp);
525         temp = (P_E_HDR *) *p;
526         if (temp->length)
527             if (temp->length < *len && strchr("tlreacdsnm", temp->type))
528                 return(temp);
529             else
530                 *p = NULL;
531     }
532     return(NULL);
533 }
534
535

```

167

5,502,839

168

```

536 Replace(area, list, msg, size, appl)
537 AREA *area;
538 LIST *list;
540 register long msg, size, appl;
541 {
542     register char *p;
543     register short length = 0;
544     register ELEMENT *temp;
545     register P E HDR *hdr, *temp_hdr = NULL;
546     register long list_len;
547     ELEMENT *after = NULL;
548
549     if (Find_triple(msg, "0\0\0\0\0", size, NO, 0, NULL))
550     {
551         Erase(area, list, msg, size, appl);
552         after = list->current;
553     }
554     if (p = Find_triple(msg, "data", size, NULL, 1, NULL))
555     {
556         list_len = *((long *) (p-4));
557         while ((length = *(short *) p) && (list_len -= length) > 0)
558         {
559             hdr = (P E HDR *) p;
560             if (hdr->type == 'm' && !check_macro(hdr))
561                 break;
562             for (temp = list->last; temp &&
563                  ((P E HDR *) &temp->length)->row != hdr->row &&
564                  ((P E HDR *) &temp->length)->col != hdr->col; temp = temp->pre);
565             if (temp)
566             {
567                 temp_hdr = (P E HDR *) &temp->length;
568                 temp->deleted = YES;
569                 after = temp->pre;
570             }
571             draw_elements(hdr, length, list, after);
572             if (temp_hdr && (hdr->type != 't' || hdr->height != temp_hdr->height
573                  || hdr->width != temp_hdr->width))
574             {
575                 change_area(area, temp_hdr->row, temp_hdr->col,
576                             temp_hdr->height, temp_hdr->width);
577                 list->erases++;
578             }
579             p += length;
580             Long_align(p);
581         }
582     }
583     if (length)
584         reply_status(msg, "--replace", "bad length/type/macro", 0);
585 }

```

169

5,502,839

170

```

586 Erase(area, list, msg, size, appl)
587 AREA *area;
588 register LIST *list;
589 register long msg, size, appl;
590 {
591     register ELEMENT *element = NULL;
592     register P_E_HDR *hdr;
593     int number;
594
595     if (element = mark_elements(list, NULL, &number, msg, size, appl))
596     {
597         list->current = element->pre;
598         for (; element; element = element->nxt)
599             if (element->marked)
600             {
601                 element->deleted = YES;
602                 hdr = (P_E_HDR*) &element->length;
603                 change_area(area, hdr->row, hdr->col, hdr->height, hdr->width);
604             }
605         list->erases += number;
606         list->changes += number;
607     }
608 }
609
610 Copy(cur, area, list, msg, size, appl)
611 CURRENT *cur;
612 register AREA *area;
613 register LIST *list;
614 register long msg, size, appl;
615 {
616     register ELEMENT *element;
617     register short bkgd, *p;
618     short *q;
619     unsigned int length = 0;
620
621     if (bkgd = (short) Find_triple(msg, "bkgd", size, NO, 0, NULL))
622     {
623         p = (short *) Find_triple(msg, "@pos", size, &none, 0, NULL);
624         q = (short *) Find_triple(msg, "@end", size, &none, 0, NULL);
625         change_area(area, *p, *(p+1), *q-*p, *(q+1)-*(p+1));
626     }
627     if ((element = mark_elements(list, &length, NULL, msg, size, appl)) || bkgd)
628         send(cur, area, list, 0, length, element, NO, NO, bkgd);
629     else
630         Reply(msg, Newmsg(64, "write", NULL));
631 }
632

```

171

5,502,839

172

```

633
634 Move(area,list,msg,size,appl)
635 AREA *area;
636 LIST *list;
637 long msg, size, appl;
638 {
639     register ELEMENT *element;
640     register P E HDR *hdr;
641     register int delta_row, delta_col, by_offset = NO, row = 0, col = 0;
642     register char *p;
643     int n;
644
645     if (p = Find_triple(msg,"by ",size,NULL,4,NULL))
646     {
647         by_offset = YES;
648         delta_row = *((short *) p)++;
649         delta_col = *((short *) p);
650     }
651     else if (p = Find_triple(msg,"to ",size,NULL,4,NULL))
652     {
653         row = *((short *) p)++;
654         col = *((short *) p);
655     }
656     if (list->current = element = mark_elements(list,NULL,&n,msg,size,appl))
657     {
658         if (!by_offset)
659         {
660             hdr = (P E HDR *) &element->length;
661             delta_row = row - hdr->row;
662             delta_col = col - hdr->col;
663         }
664         for (; element; element = element->nxt)
665             if (element->marked)
666             {
667                 hdr = (P E HDR *) &element->length;
668                 change_area(area,hdr->row,hdr->col,hdr->height,hdr->width);
669                 hdr->row += delta_row;
670                 hdr->col += delta_col;
671                 element->changed = YES;
672                 element->marked = NO;
673                 element->deleted = (hdr->row < 0 || hdr->col < 0);
674             }
675         list->changes += n;
676         list->erases += n;
677     }
678 }

```

173

5,502,839

174

```

679 Change(area,list,msg,size,appl)
680 register AREA *area;
681 register LIST *list;
682 register long msg, size, appl;
683 {
684     register ELEMENT *element = NULL;
685     register P_E_HDR *hdr;
686     char *color, *bkgd, *fill, *pat;
687
688     color = Find triple(msg,"colr",size,NULL,1,NULL);
689     bkgd = Find triple(msg,"bkgd",size,NULL,1,NULL);
690     fill = Find triple(msg,"fill",size,NULL,1,NULL);
691     pat = Find triple(msg,"pat",size,NULL,1,NULL);
692     if (list->current = element = mark elements(list,NULL,NULL,msg,size,appl))
693         for ( ; element; element = element->nxt)
694             if (element->marked)
695                 {
696                     hdr = (P E HDR*) &element->length;
697                     if (color)
698                         hdr->color = *color;
699                     if (bkgd)
700                         hdr->bkgrnd = *bkgd;
701                     if (fill)
702                         hdr->fill = *fill;
703                     if (pat)
704                         hdr->pattern = *pat;
705                     change area(area,hdr->row,hdr->col,hdr->height,hdr->width);
706                     list->changes++;
707                 }
708 }
709
710 Background(area,list,msg,size)
711 register AREA *area;
712 register LIST *list;
713 register long msg, size;
714 {
715     area->color = *Find triple(msg,"colr",size,&area->color,1,NULL);
716     area->pattern = *Find triple(msg,"pat",size,&area->pattern,1,NULL);
717     change area(area,0,0,MAX ROW,MAX COL);
718     list->changes = list->erases = 1;
719 }
720

```

175

5,502,839

176

```

721 New picture(cur, area, list)
722 register CURRENT *cur;
723 register AREA *area;
724 register LIST *list;
725
726 {
727     register ELEMENT *element;
728     register long max, maxe;
729     short def_max = 20, def_maxe = 100;
730     char def_bkgd = BLACK, def_pat = 0;
731
732     for (element = list->first; element; element = element->nxt)
733         element->deleted = YES;
734     list->current = list->first = list->last = NULL;
735     list->changes = list->erases = list->size = 0;
736     if (Find_triple(cur->msg, "file", cur->size, NO, 0, NULL))
737         return(Old_picture(cur, list));
738     else
739     {
740         cur->owner = cur->sender;
741         strcpy(cur->name, Find_triple(cur->msg, "name", cur->size, &none, 1, NULL));
742         area->max_height =
743             *(short*)Find_triple(cur->msg, "size", cur->size, &none, 4, NULL);
744         area->max_width =
745             *(short*) (Find_triple(cur->msg, "size", cur->size, &none, 4, NULL)+2);
746         area->color = *Find_triple(cur->msg, "bkgd", cur->size, &def_bkgd, 1, NULL);
747         area->pattern = *Find_triple(cur->msg, "pat", cur->size, &def_pat, 1, NULL);
748         cur->highlight = *Find_triple(cur->msg, "high", cur->size, &none, 1, NULL);
749         cur->check = (area->max_height != 0);
750         max =
751             *(short*)Find_triple(cur->msg, "max ", cur->size, &def_max, 2, NULL)+1;
752         maxe = *(short*)Find_triple(cur->msg, "maxe", cur->size, &def_maxe, 2, NULL);
753         if (maxe & 1)
754             ++maxe;
755         list->pool.n = max;
756         list->pool.size = maxe + sizeof(ELEMENT) + 10;
757         list->pool.ptr = (ELEMENT *) Alloc(max*list->pool.size, YES);
758         memset(list->pool.ptr, 0, max*list->pool.size);
759         change_area(area, 0, 0, MAX_ROW, MAX_COL);
760         list->changes = list->erases = 1;
761         reply_status(cur->msg, "+create", "complete", 0);
762         return(YES);
763     }
764 }

```

177

5,502,839

178

```

765 Old picture(cur_list)
766 register CURRENT *cur;
767 register LIST *list;
768
769 {
770     register char *p = (char*)1;
771     CONNECTOR file;
772
773     strcpy(cur->name, Find_triple(cur->msg, "name", cur->size, &none, 1, NULL));
774     strcpy(cur->file, Find_triple(cur->msg, "file", cur->size, cur->name, 1, NULL));
775     if (*cur->file)
776     {
777         if (Connect to(NEXT "File mgt" Newmsg(64, "open"
778             "name=#S; omod=#S; amod=#S", cur->file, "R", NULL), &file))
779         {
780             cur->owner = cur->sender;
781             while (p)
782                 if (p = Call(DIRECT, file.pid,
783                     Newmsg(64, "read", "conn=#C; size=#1", &file, -1), 0, 0))
784                     if (p = Find_triple(p, "data", 0, NULL, 4, NULL))
785                         draw_elements(p, *(long*)(p-4), list, NULL);
786             Put(DIRECT, file.pid, Newmsg(32, "close", "conn=#C", &file));
787             reply_status(cur->msg, "+open", "complete", 0);
788             return(YES);
789         }
790         else
791             reply_status(cur->msg, "-open", "can't open file", 0);
792     }
793     else
794         reply_status(cur->msg, "-open", "no file name", 0);
795     return(NO);
796 }

```

179

5,502,839

180


```

797
798 Save_picture(cur,list)
799 CURRENT *cur;
800 LIST *list;
801 (
802     register char *file_name, *m, *p;
803     register ELEMENT *element;
804     CONNECTOR file;
805     unsigned int length = 0, num;
806
807     if (!(file_name = Find_triple(cur->msg, "file", cur->size, NULL, 1, NULL)))
808         file_name = cur->file;
809     if (*file_name)
810         if (element =
811             mark_elements(list, &length, &num, cur->msg, cur->size, cur->appl))
812         {
813             if (!Connect_to(NEXT, "File mgt", Newmsg(64, "open",
814                 "name=#S; omod=#S; amod=#S", file_name, "W", NULL), &file))
815                 Connect_to(NEXT, "File mgt", Newmsg(64, "create",
816                     "name=#S; omod=#S; amod=#S", file_name, "W", NULL), &file);
817             if (file.pid)
818             {
819                 num = length + 4 * num + 4;
820                 m = Newmsg(num+50, "write", "conn=#C; data=#A", &file, num, NULL);
821                 p = Find_triple(m, "data", 0, NULL, 1, NULL);
822                 for (; element; element = element->nxt)
823                     if (element->marked)
824                     {
825                         memcpy(p, element, element->length);
826                         p += element->length;
827                         Long_align(p);
828                     }
829                 *(short *) p = NULL;
830                 Put(DIRECT, file.pid, m);
831                 Put(DIRECT, file.pid, Newmsg(32, "close", "conn=#C", &file));
832                 reply_status(cur->msg, "+save", "picture saved", 0);
833             }
834             else
835                 reply_status(cur->msg, "-save", "can't open/create file", 0);
836         }
837         else
838             reply_status(cur->msg, "-save", "no elements", 0);
839     else
840         reply_status(cur->msg, "-save", "no file name", 0);
841 )

```

181

5,502,839

182

```

842
843 Appl(cur,list)
844 CURRENT *cur;
845 register LIST *list;
846 {
847     register APPL *appl;
848     register long name;
849     register short *p;
850
851     name = *(long *) Find_triple(cur->msg,"name",cur->size,&none,4,NULL);
852     for (appl = list->appls; appl && appl->name != name; appl = appl->nxt);
853     if (!appl)
854         appl = (APPL *) Alloc(sizeof(APPL),YES);
855     appl->conn = cur->sender;
856     p = (short *) Find_triple(cur->msg,"org ",cur->size,&none,2,NULL);
857     appl->row = *p++;
858     appl->col = *p;
859     appl->name = name;
860     appl->conn =
861     *(CONNECTOR *) Find_triple(cur->msg,"appl",cur->size,&none,4,NULL);
862     appl->nxt = list->appls;
863     list->appls = appl;
864 }
865
866 Move mark(cur,area,list)
867 register CURRENT *cur;
868 register AREA *area;
869 register LIST *list;
870 {
871     register P E HDR *hdr;
872     register short *pos;
873     char *q;
874
875     if (pos = (short *) Find_triple(cur->msg,"at ",cur->size,NULL,4,NULL))
876     {
877         if (cur->mark)
878             erase_mark(cur,area);
879         else
880         {
881             q = cur->mark = Alloc(sizeof(P E HDR)+30,YES);
882             draw_line(&q,0,0,VCHAR_HT,0,NULL,YELLOW,'S',0,1,NULL);
883         }
884         hdr = (P E HDR *) cur->mark;
885         hdr->row = *pos++ - ((hdr->height - VCHAR_HT) / 2);
886         hdr->col = *pos;
887         cur->display_mark = YES;
888         list->changes++;
889     }
890 }

```

183

5,502,839

184

```

891
892 Query mark(cur)
893 register CURRENT *cur;
894 {
895     register P_E_HDR *hdr;
896
897     if (hdr = (P_E_HDR *) cur->mark)
898         Reply(cur->msg, Newmsg(64, "mark", "at=#2s", hdr->row, hdr->col));
899     else
900         reply_status(cur->msg, "--mark?", "no mark defined", 0);
901 }
902
903 Set mark(cur, area, list)
904 register CURRENT *cur;
905 register AREA *area;
906 register LIST *list;
907 {
908     register P_E_HDR *hdr;
909
910     if ((hdr = (P_E_HDR *) Find_triple(cur->msg, "data", cur->size, NULL, 1, NULL))
911         && hdr->length)
912     {
913         if (cur->mark)
914         {
915             erase mark(cur, area);
916             Free(cur->mark);
917             Free(cur->erase_mark);
918             cur->erase_mark = NULL;
919         }
920         cur->mark = Alloc(hdr->length, YES);
921         memcpy(cur->mark, hdr, hdr->length);
922         cur->display_mark = YES;
923     }
924     else
925     {
926         if (cur->old mark)
927             Free(cur->old mark);
928         cur->old mark = cur->mark;
929         cur->mark = NULL;
930     }
931     list->changes++;
932 }

```

185

5,502,839

186

```

933
934 Restore mark(cur,area,list)
935 register CURRENT *cur;
936 register AREA *area;
937 register LIST *list;
938 {
939     if (cur->old_mark)
940     {
941         if (cur->mark)
942         {
943             erase mark(cur,area);
944             Free(cur->mark);
945             Free(cur->erase mark);
946             cur->erase_mark = NULL;
947         }
948         cur->mark = cur->old mark;
949         cur->old mark = NULL;
950         list->changes++;
951     }
952 }
953
954 erase mark(cur,area)
955 register CURRENT *cur;
956 register AREA *area;
957 {
958     if (!cur->erase mark)
959         cur->erase mark = Alloc(*(short*)cur->mark,YES);
960     memcpy(cur->erase mark,cur->mark,*(short*)cur->mark);
961     ((P_E_HDR *)cur->erase_mark)->color = area->color;
962 }
963
964 Edit_text(cur,area,list,msg,size,appl)
965 CURRENT *cur;
966 AREA *area;
967 register LIST *list;
968 register long msg, size;
969 long appl;
970 {
971

```

187

5,502,839

188

```

972 register char *p, c, *text_start, *new;
973 register short shift, offset, sel_offset, ok = YES;
974 short sel_length;
975 ELEMENT *element;
976 P_E_HDR *hdr;
977
978 if (list->current = element = mark_elements(list, NULL, NULL, msg, size, appl))
979 {
980     offset = *(short *) Find_once(msg, "offs", size, &none, 2, NULL);
981     hdr = (P_E_HDR *) element->length;
982     if (hdr->type == 't')
983     {
984         text_start = (p = value(hdr) + sizeof(long)) + 2 * sizeof(short);
985         if (shift = *(short *) Find_once(msg, "shift", size, &none, 2, 0))
986             shift_text(p, text_start, shift);
987         if (Find_once(msg, "set ", size, NO, 0, NULL))
988         {
989             sel_offset = *((short *) p)++;
990             sel_length = *(short *) p;
991             ok = (offset < sel_length);
992             offset += sel_offset;
993         }
994         if (ok = (ok && (offset < strlen(text_start))))
995         {
996             p = text_start + offset;
997             if (new = Find_once(msg, "new ", size, NULL, 1, NULL))
998             {
999                 while (c = *new++)
1000                     if (c > 31 && c < 127 && *p)
1001                         *p++ = c;
1002                     else if (c == 8 && p > text_start)
1003                         *--p = ',';
1004             }
1005             if (Find_once(msg, "blk", size, NO, 0, NULL))
1006                 for (; *p; *p++ = ' ');
1007             if (Find_triple(msg, "by ", size, NO, 0, NULL))
1008             {
1009                 Move(area, list, msg, size, appl);
1010                 Draw(list, msg, size);
1011             }
1012             else
1013             {
1014                 element->changed = YES;
1015                 list->changes++;
1016             }
1017             Move_mark(cur, area, list);
1018             if (Find_once(msg, "fast", size, NO, 0, NULL))
1019                 list->erases = 0;

```

189

5,502,839

190

```

1020         )
1021         else
1022             reply_status(msg, "-edit", "outside text string", 0);
1023     )
1024     else
1025         reply_status(msg, "-edit", "not a text element", 0);
1026 )
1027 else
1028     reply_status(msg, "-edit", "not found", 0);
1029 )
1030
1031 shift text(sel, text, nchars)
1032 register short *sel, nchars;
1033 register char *text;
1034 {
1035     register short length, n;
1036
1037     if (length = strlen(text))
1038     {
1039         if (nchars < 0 && (n = length + nchars) > 0)
1040         {
1041             memcpy(text, text+n, -nchars);
1042             memset(text-nchars, ' ', n);
1043             if (*sel - n >= 0)
1044                 *sel -= n;
1045             else
1046             {
1047                 *sel = 0;
1048                 *(sel+1) += *sel - n;
1049             }
1050         }
1051         else if (nchars > 0 && (n = length - nchars) > 0)
1052         {
1053             memcpy(text+length, text+nchars, nchars);
1054             memset(text, ' ', n);
1055             if (*sel + n < length)
1056                 *sel += n;
1057             else
1058             {
1059                 *sel = length - n;
1060                 *(sel+1) -= *sel + n - length;
1061             }
1062         }
1063     }

```

191

5,502,839

192

```

1063
1064 Animate(cur, list)
1065 register CURRENT *cur;
1066 register LIST *list;
1067 {
1068     register ANIM *anim;
1069     register char *name;
1070     register long pid;
1071     char *m;
1072
1073     if (name = Find triple(cur->msg, "name", cur->size, NULL, 2, NULL))
1074     {
1075         if (strlen(name) < 16)
1076         {
1077             for (anim = list->anims; anim && strcmp(name, anim->name);
1078                 anim = anim->nxt);
1079             if (!anim)
1080             {
1081                 if (pid = NewProc(name, "//processes/animate", YES, -1))
1082                 {
1083                     anim = (ANIM *) Alloc(sizeof(ANIM), YES);
1084                     anim->conn.pid = pid;
1085                     strcpy(anim->name, name);
1086                     anim->nxt = list->anims;
1087                     list->anims = anim;
1088                     m = Alloc(cur->size, YES);
1089                     memcpy(m, cur->msg, cur->size);
1090                     Put(DIRECT, anim->conn.pid, m);
1091                 }
1092             }
1093             else
1094                 reply_status(cur->msg, "-animate", "not supported", 0);
1095         }
1096         else
1097             reply_status(cur->msg, "-animate", "duplicate name", 0);
1098     }
1099     else
1100         reply_status(cur->msg, "-animate", "name too long", 0);
1101 }

```

193

5,502,839

194

```

1098
1099 Alter(cur,list)
1100 register CURRENT      *cur;
1101 register LIST         *list;
1102 {
1103     register ANIM      *anim;
1104     register char       *name;
1105     CONNECTOR          conn;
1106
1107     if (name = Find_triple(cur->msg,"name",cur->size,NULL,2,NULL))
1108     {
1109         for (anim = list->anims; anim && strcmp(name,anim->name);
1110             anim = anim->nxt);
1111         if (anim)
1112         {
1113             conn = anim->conn;
1114             if (!strcmp(cur->msg,"cancel"))
1115             {
1116                 list->anims = anim->nxt;
1117                 Free(anim);
1118             }
1119             Forward(DIRECT,conn.pid,cur->msg);
1120             cur->msg = NULL;
1121         }
1122         else
1123             reply_status(cur->msg,cur->msg,"not found",0);
1124     }
1125 }

```

195

5,502,839

196


```

1126 hit(list,msg,size,appl)
1127 register LIST *list;
1128 register long msg, size, appl;
1129 {
1130     register short *p, tolerance;
1131     register ELEMENT *element;
1132     register P_E_HDR *hdr;
1133     ELEMENT *find_box();
1134     tolerance = *(short *) Find_triple(msg,"tolr",size,&none,2,NULL);
1135     if (p = (short *) Find_triple(msg,"pos",size,&none,4,NULL))
1136     {
1137         if (list->current = element = find_box(*p,*(p+1),list,appl))
1138         {
1139             hdr = (P_E_HDR *) &element->length;
1140             if (Find_triple(msg,"sel",size,NO,0,NULL) && hdr->attr.selectable)
1141             {
1142                 hdr->attr.selected = YES;
1143                 if ((hdr->type == 'm') && (*value(hdr) == 'L'))
1144                     sel_list(hdr);
1145                 element->changed = YES;
1146                 list->changes++;
1147             }
1148             Reply(msg,Newmsg(hdr->length+50,"write","data=#e#e",hdr,NULL));
1149         }
1150     }
1151     else
1152         reply_status(msg,msg,"not found",0);
1153     else
1154         reply_status(msg,msg,"missing \'pos\'",0);
1155 }

```

197

5,502,839

198

```

1156 ELEMENT *find_box(row,col,list,appl)
1157 register short row,col;
1158 register LIST *list;
1159 register long appl;
1160 {
1161     register P_E_HDR *hdr;
1162     register ELEMENT *element;
1163     for (element = list->last; element; element = element->pre)
1164     {
1165         hdr = (P_E_HDR *) &element->length;
1166         if (in_box(hdr->row, hdr->col, hdr->height, hdr->width, row, col)
1167             && !element->deleted)
1168             if (!appl || (appl == -1 && !*(long*)(hdr+1))
1169                 || appl == *(long*)(hdr+1))
1170                 break;
1171     }
1172     return(element);
1173 }
1174
1175 in_box(r,c,h,w,c1,c2)
1176 register short r, c, c1, c2, h, w;
1177 {
1178     if ((c1 < r) || (c2 < c))
1179         return(NO);
1180     if ((c1 > r + h) || (c2 > c + w))
1181         return(NO);
1182     return(YES);
1183 }
1184
1185 sel_list(hdr)
1186 register P_E_HDR *hdr;
1187 {
1188     register P_E_HDR *temp, *first;
1189     short len;
1190     char *p;
1191     for (first = temp = first_macro(hdr, NULL, &len, &p);
1192         temp && temp->attr.hidden; temp = next_macro(&len, &p));
1193     if (temp)
1194     {
1195         temp->attr.hidden = YES;
1196         if (!(temp = next_macro(&len, &p)))
1197             temp = first;
1198         temp->attr.hidden = NO;
1199     }
1200 }
1201
1202
1203

```

199

5,502,839

200

```

1204 Viewport(cur,area,list)
1205 register CURRENT *cur;
1206 register AREA *area;
1207 register LIST *list;
1208
1209 {
1210     register VIEW *view, *prev = NULL;
1211     CONNECTOR *conn;
1212     ELEMENT *element;
1213     unsigned int length = 0;
1214     char *p;
1215
1216     if (p = Find_triple(cur->msg,"area",cur->size,NULL,8,NULL))
1217     {
1218         for (view = list->views; view && (view->owner.pid != cur->sender.pid);
1219             view = view->nxt) ;
1220         if (view)
1221             memcpy(&view->row,p,4*sizeof(short));
1222         else
1223         {
1224             view = (VIEW *) Alloc(sizeof(VIEW),YES);
1225             view->nxt = list->views;
1226             view->owner = cur->sender;
1227             memcpy(&view->row,p,4*sizeof(short));
1228             list->views = view;
1229         }
1230         change_area(area,view->row,view->col,view->height,view->width);
1231         element = mark_area(area->r1,area->c1,area->r2,area->c2,list,
1232             MAX_P_E,NULL,NULL,NULL,&length,NULL,cur->appl);
1233         send(cur,area,list,0,length,element,YES,cur->display_mark,YES);
1234     }
1235     else
1236     {
1237         conn = (CONNECTOR *) Find_triple(cur->msg,"conn",0,&cur->sender,8,NULL);
1238         for (view = list->views; view && (view->owner.pid != conn->pid);
1239             prev = view, view = view->nxt) ;
1240         if (view)
1241         {
1242             if (prev)
1243                 prev->nxt = view->nxt;
1244             else
1245                 list->views = view->nxt;
1246             Free(view);
1247         }
1248     }
1249 }

```

201

5,502,839

202

```

1250 change area(area,row,col,height,width)
1251 register AREA *area;
1252 register short row,col,height,width;
1253 {
1254     if (row < area->r1)
1255         area->r1 = row;
1256     if (col < area->c1)
1257         area->c1 = col;
1258     if (row + height > area->r2)
1259         area->r2 = row + height;
1260     if (col + width > area->c2)
1261         area->c2 = col + width;
1262 }
1263
1264 notify(cur,area,list)
1265 register CURRENT *cur;
1266 register AREA *area;
1267 LIST *list;
1268 {
1269     register VIEW *view;
1270     register int length;
1271     for (view = list->views; view; view = view->nxt)
1272     {
1273         length = mark_changes(list->first,
1274             view->row,view->col,view->height,view->width);
1275         send(cur,area,list,&view->owner,length,list->first,
1276             YES,cur->display_mark,list->erases);
1277     }
1278 }
1279
1280 mark_changes(element,r,c,h,w)
1281 register ELEMENT *element;
1282 register short r,c,h,w;
1283 {
1284     register P E HDR *hdr;
1285     register int list_length = 0;
1286     for ( ; element && !element->changed; element = element->nxt) ;
1287     for ( ; element; element = element->nxt)
1288     {
1289         hdr = (P E HDR *) &element->length;
1290         if (element->marked = (element->changed && !hdr->attr.hidden &&
1291             (hdr->row + hdr->height >= r) && (hdr->row <= r + h) &&
1292             (hdr->col + hdr->width >= c) && (hdr->col <= c + w)))
1293             list_length += hdr->length + 3;
1294     }
1295     return(list_length);
1296 }
1297
1298
1299

```

203

5,502,839

204

```

1300
1301 send(cur,area,list,proc,length,element,modify,mark,redraw)
1302 register CURRENT *cur;
1303 AREA *area;
1304 LIST *list;
1305 register CONNECTOR *proc;
1306 register unsigned int length;
1307 register ELEMENT *element;
1308 register unsigned short modify, mark, redraw;
1309 {
1310     register P E HDR *hdr;
1311     register short element_length;
1312     char *m,*p,*set mark();
1313     ELEMENT *redraw_bkgd();
1314
1315     if (redraw)
1316         element = redraw_bkgd(area,list,&m,&p);
1317     else
1318         p = (m = Newmsg(length+300,
1319             "write","data=#A; type=#c",length+250,NULL,'P')) + 24;
1320     if (element)
1321         for ( ; element; element = element->nxt)
1322         {
1323             if (element->marked && !element->deleted)
1324             {
1325                 element->marked = 0;
1326                 element_length = element->length;
1327                 memcpy(p,&element->length,(long)element_length);
1328                 hdr = (P E HDR *) p;
1329                 if (modify)
1330                 {
1331                     if (hdr->attr.selected)
1332                         element_length = set_select(hdr,cur->highlight);
1333                     if ((hdr->type == 'm') && (*value(hdr) == 'L'))
1334                         element_length = macro_list(hdr);
1335                     if ((hdr->type == 't'))
1336                         element_length = check_text(hdr,hdr->length);
1337                     if (cur->appl)
1338                         element_length =
1339                             change_origin(hdr,cur->appl_row,cur->appl_col);
1340                 }
1341                 p += element_length;
1342                 Long_align(p);
1343             }
1344         }
1345     if (mark)
1346         p = set mark(p,cur);
1347     *(short *) p = NULL;
1348     if (proc)
1349         Put(DIRECT,proc->pid,m);
1350     else
1351         Reply(cur->msg,m);
1352 }

```

205

5,502,839

206

```

1353 change origin(hdr,row,col)
1354 register P E HDR *hdr;
1355 register short row, col;
1356 {
1357     if ((hdr->row == row) < 0)
1358         return(0);
1359     if ((hdr->col == col) < 0)
1360         return(0);
1361     return(hdr->length);
1362 }
1363
1364 char *set mark(p,cur)
1365 register char *p;
1366 register CURRENT *cur;
1367 {
1368     if (cur->erase_mark)
1369     {
1370         memcpy(p,cur->erase_mark,*(short*)cur->erase_mark);
1371         p += *(short *) p;
1372     }
1373     if (cur->mark)
1374     {
1375         memcpy(p,cur->mark,*(short*)cur->mark);
1376         p += *(short *) p;
1377     }
1378     return(p);
1379 }
1380
1381 ELEMENT *redraw bkgd(area,list,buf,ptr)
1382 register AREA *area;
1383 register LIST *list;
1384 register char **buf, **ptr;
1385 {
1386     ELEMENT *element;
1387     int length, num;
1388
1389     element = mark area(area->r1,area->c1,area->r2,area->c2,
1390         list,MAX P E,NULL,NULL,NULL,&length,&num,NULL);
1391     length += (4 * num) + 150;
1392     *buf = Newmsg(length+50,"write","data=#A; type=#c",length,NULL,'P');
1393     *ptr = *buf + 24;
1394     draw filled rect(ptr,area->r1,area->c1,(area->r2)-(area->r1),
1395         (area->c2)-(area->c1),NULL,0,0,area->color,area->pattern,0,0,0,NULL);
1396     return(element);
1397 }
1398

```

207

5,502,839

208

```

1399 set select(hdr,high_option)
1400 register P_E_HDR *hdr;
1401 register char high_option;
1402 {
1403     register short length;
1404     length = hdr->length;
1405     if (!high_option)
1406         hdr->attr.invert = !hdr->attr.invert;
1407     else if (high_option == 'I')
1408         hdr->attr.invert = !hdr->attr.invert;
1409     else if (high_option == 'H')
1410         hdr->attr.highlight = !hdr->attr.highlight;
1411     else if (high_option == 'C')
1412     {
1413         if (hdr->type != 'm')
1414         {
1415             hdr->color = (hdr->color + 1) % 7 + 1;
1416             if (hdr->fill)
1417                 hdr->fill = (hdr->fill + 1) % 7 + 1;
1418         }
1419         else
1420             macro_color(hdr);
1421     }
1422     else if (hdr->type == 't')
1423         sel_text(hdr,high_option);
1424     return(length);
1425 }
1426
1427 macro list(hdr)
1428 register P_E_HDR *hdr;
1429 {
1430     register P_E_HDR *temp;
1431     register short row,col;
1432     short len;
1433     char *p;
1434
1435     row = hdr->row;
1436     col = hdr->col;
1437     for (temp = first_macro(hdr,NULL,&len,&p);
1438          temp && temp->attr.hidden; temp = next_macro(&len,&p));
1439     if (temp)
1440     {
1441         memcpy(hdr,temp,temp->length);
1442         hdr->row = row;
1443         hdr->col = col;
1444     }
1445     return(hdr->length);
1446 }
1447
1448

```

209

5,502,839

210

```

1449 macro color(hdr)
1450 register P_E_HDR *hdr;
1451 {
1452     register P_E_HDR *temp;
1453     short len;
1454     char *p;
1455     for (temp = first_macro(hdr, NULL, &len, &p); temp; temp = next_macro(&len, &p))
1456     {
1457         temp->color = (temp->color + 1) % 7 + 1;
1458         if (temp->fill)
1459             temp->fill = (temp->fill + 1) % 7 + 1;
1460     }
1461 }
1462
1463 sel_text(hdr, high_option)
1464 register P_E_HDR *hdr;
1465 register char high_option;
1466 {
1467     register TEXT_OPTIONS *opt;
1468     opt = (TEXT_OPTIONS *) value(hdr);
1469     if (high_option == 'b')
1470         opt->border = YES;
1471     else if (high_option == 'U')
1472         opt->underline = YES;
1473     else if (high_option == 'B')
1474         opt->bold = YES;
1475 }
1476
1477 check_text(hdr, length)
1478 register P_E_HDR *hdr;
1479 register short length;
1480 {
1481     register char *p;
1482     char *h;
1483     register TEXT_OPTIONS *opt;
1484     opt = (TEXT_OPTIONS *) value(hdr);
1485     if (opt->border && hdr->fill)
1486     {
1487         opt->border = NO;
1488         p = (char *) hdr + length;
1489         long align(p);
1490         n = p;
1491         draw_rect(&n, hdr->row-3, hdr->col-3, hdr->height+6, hdr->width+6,
1492                 NULL, hdr->fill, 'S', 1, NULL);
1493         length = n - (char *) hdr;
1494     }
1495     return(length);
1496 }
1497
1498
1499
1500

```

211

5,502,839

212


```

1501
1502 ELEMENT *mark_elements(list, length, num, msg, size, appl)
1503     *list;
1504 unsigned int     *length, *num;
1505 long             msg, size, appl;
1506 {
1507     register short row = 0, col = 0, number = 0, count = 1;
1508     register short to_row = MAX_ROW, to_col = MAX_COL, *p;
1509     register ELEMENT *element;
1510     register char *tag_pat = NULL;
1511     char what = NULL, tag_buf[200], *text_pat = NULL, dflt = YES;
1512     long *triple, attr = NULL;
1513
1514     element = NULL;
1515     while (p = (short*)Find_triple(msg, "\0\0\0\0", size, NULL, 0, &triple))
1516     {
1517         switch (*triple)
1518         {
1519             case Keypack('e', 'c', 'n', 't'): count = *p;
1520             break;
1521             case Keypack('e', 'a', 't', 't'): attr = *(long *) p;
1522             break;
1523             case Keypack('e', 's', 'e', 'l'): attr = 0x8000;
1524             break;
1525             case Keypack('e', 'n', 'u', 'm'): number = *p;
1526             what = NULL;
1527             break;
1528             case Keypack('e', 'p', 'o', 's'): row = *p++;
1529             col = *p;
1530             what = 'A';
1531             break;
1532             case Keypack('e', 'e', 'n', 'd'): to_row = *p++;
1533             to_col = *p;
1534             what = 'A';
1535             break;
1536             case Keypack('e', 't', 'x', 't'): text_pat = Alloc(500, YES);
1537             if (!makpat(p, text_pat))
1538             {
1539                 Free(text_pat);
1540                 text_pat = NULL;
1541             }
1542             break;
1543             case Keypack('e', 't', 'a', 'g'): if (!makpat(p, (tag_pat = tag_buf)))
1544             tag_pat = NULL;
1545             break;
1546         }
1547         *triple = NULL;
1548         dflt = NO;
1549     }
1550     if (dflt)
1551         count = MAX_P_E;
1552     if (!what)
1553         element = mark_number(number, tag_pat, text_pat,
1554                                list, count, attr, length, num, appl);
1555     else if (what == 'A')
1556         element = mark_area(row, col, to_row, to_col, list, count,
1557                             attr, tag_pat, text_pat, length, num, appl);

```

213

5,502,839

214

```

1557     if (text_pat)
1558         freetext_pat);
1559     return(element);
1560 }
1561
1562 ELEMENT *mark_area(row,col,to_row,to_col,list
1563                   count,attr,tag_pat,text_pat,length,num,appl)
1564 register short    row,col,to_row,to_col,count;
1565 long             attr;
1566 LIST             *list;
1567 char             *tag_pat,*text_pat;
1568 unsigned int     *length,*num;
1569 {
1570     register P E HDR *hdr;
1571     register ELEMENT *element = NULL,*temp;
1572     register long    total_length = 0;
1573     unsigned int     orig_count;
1574
1575     if (row >= 0 && col >= 0 && to_row >= 0 && to_col >= 0)
1576     {
1577         orig_count = count;
1578         for (temp = list->first; temp && count; temp = temp->nxt)
1579         {
1580             hdr = (P E HDR *) &temp->length;
1581             if (hdr->row <= to_row && hdr->col <= to_col
1582                 && row < hdr->row + hdr->height && col < hdr->col + hdr->width
1583                 && valid(hdr,tag_pat,text_pat,attr,appl) && !temp->deleted)
1584             {
1585                 total_length += temp->length;
1586                 temp->marked = YES;
1587                 if (!element)
1588                     element = temp;
1589                 count--;
1590             }
1591         }
1592         if (length)
1593             *length = total_length;
1594         if (num)
1595             *num = orig_count - count;
1596     }
1597     return(element);
1598 }

```

215

5,502,839

216

```

1599 ELEMENT *mark number(n,tag_pat,text_pat,list,count,attr,length,num,appl)
1600 register short      n, count;
1601 register long       tag_pat, text_pat, attr;
1602 register LIST       *list;
1603 unsigned int        *length, *num;
1604 {
1605     register ELEMENT *element = NULL, *temp = NULL;
1606     register long     total_length = 0;
1607     unsigned int      orig_count;
1608     if (n == -1)
1609         temp = list->last;
1610     else
1611         for (temp = list->first; temp && n--; temp = temp->nxt);
1612     for (orig_count = count; temp && count; temp = temp->nxt)
1613         if (valid(&temp->length,tag_pat,text_pat,attr,appl) && !temp->deleted)
1614         {
1615             total_length += temp->length;
1616             temp->marked = YES;
1617             if (!element)
1618                 element = temp;
1619             count--;
1620         }
1621     if (length)
1622         *length = total_length;
1623     if (num)
1624         *num = orig_count - count;
1625     return(element);
1626 }
1627

```

217

5,502,839

218

```

1629 valid(hdr, tag_pat, text_pat, attr, appl)
1630 register p E HDR *hdr;
1631 register char *tag_pat, *text_pat;
1632 register long attr, appl;
1633 {
1634     register char *target, ok = YES;
1635     long temp;
1636     if (tag_pat)
1637         if (target = tag(hdr))
1638             ok = amatch(target, tag_pat);
1639         else
1640             ok = NO;
1641     if (text_pat)
1642         if (hdr->type == 't')
1643             ok = ok && amatch(value(hdr)+8, text_pat);
1644         else
1645             ok = NO;
1646     if (attr)
1647     {
1648         memcpy(&temp, &hdr->attr, sizeof(long));
1649         ok = ok && (temp & attr);
1650     }
1651     if (appl)
1652     {
1653         if (appl == -1)
1654             ok = ok && (!*(long*)(hdr+1));
1655         else
1656             ok = ok && (appl == *(long*)(hdr+1));
1657     }
1658     return(ok);
1659 }
1660
1661
1662 status(msg, size)
1663 register char *msg;
1664 register long size;
1665 {
1666     register char *m;
1667     *(m = Alloc(size, YES)) = NULL;
1668     strcat(m, Find triple(msg, "orig", size, &none, 1, NULL));
1669     strcat(m, " ");
1670     strcat(m, Find triple(msg, "stat", size, &none, 1, NULL));
1671     strcat(m, " in-");
1672     strcat(m, Find triple(msg, "req ", size, &none, 1, NULL));
1673     Note(m, "ERROR");
1674     Free(m);
1675 }
1676
1677

```

219

5,502,839

220

```

1678 reply status(cur,mid,stat,code)
1679 register char *cur, *mid, *stat;
1680 register long *code;
1681 {
1682     register char *type;
1683     type = "failed";
1684     if (*mid == '-')
1685         mid++;
1686     else if (*mid == '+')
1687     {
1688         type = "done";
1689         mid++;
1690     }
1691     Reply(cur, Newmsg(strlen(mid)+strlen(stat)+50, type,
1692         "orig=#S; req=#S; stat=#S; code=#l", "picture", mid, stat, code));
1693 }
1694
1695 ELEMENT *new element(list, size, after)
1696 register LIST *list;
1697 register long size;
1698 register ELEMENT *after;
1699 {
1700     register ELEMENT *element;
1701     register long i = 0;
1702     if (size <= list->pool.size)
1703     for (i = list->pool.n, element = list->pool.ptr;
1704         element->pool && i && element->deleted;
1705         (char*)element += list->pool.size, --i);
1706     if (i)
1707     {
1708         if (element->deleted)
1709             delete element(list, element);
1710         element->pool = YES;
1711     }
1712     else
1713     {
1714         element = (ELEMENT *) Alloc(size, YES);
1715         element->pool = NO;
1716     }
1717     element->nxt = NULL;
1718     if (element->pre = list->last)
1719         (list->last)->nxt = element;
1720     else
1721         list->first = element;
1722     list->last = element;
1723     element->changed = YES;
1724     element->deleted = element->marked = NO;
1725     return(element);
1726 }
1727
1728
1729

```

221

5,502,839

222

```

1730 delete element(list,element)
1731 register ELEMENT *element;
1732 register LIST *list;
1733 {
1734     if (element->pre)
1735         (element->pre)->nxt = element->nxt;
1736     else
1737         list->first = element->nxt;
1738     if (element->nxt)
1739         (element->nxt)->pre = element->pre;
1740     else
1741         list->last = element->pre;
1742     if (element->pool)
1743         element->pool = NULL;
1744     else
1745         Free(element);
1746     --list->size;
1747 }
1748
1749 char *value(hdr)
1750 register P_E_HDR *hdr;
1751 {
1752     register char *p;
1753     p = (char *) hdr + sizeof(P_E_HDR);
1754     if (hdr->attr.appl)
1755         p += 4;
1756     if (hdr->attr.tagged)
1757         while (*p++);
1758     Long align(p);
1759     return(p);
1760 }
1761
1762 char *tag(hdr)
1763 register P_E_HDR *hdr;
1764 {
1765     register char *p;
1766     p = (char *) hdr + sizeof(P_E_HDR);
1767     if (hdr->attr.appl)
1768         p += 4;
1769     if (hdr->attr.tagged)
1770         return(p);
1771     return(NULL);
1772 }
1773

```

223

5,502,839

224

225

What is claimed is:

1. A virtual input interface in a data processing system, said interface comprising:

means for accepting input from at least one physical device and for converting said physical device input into virtual input, said means comprising a virtual input manager process responsive to said at least one physical input device for generating a picture, said picture comprising one or more picture elements, each picture element comprising a plurality of device-independent data structures in a predetermined, standard data format, at least one of said data structures comprising a plurality of different data fields each containing information describing said picture element; and

means responsive to said virtual input for performing processing operations upon said virtual input, said means comprising a console manager process for performing processing operations on said one or more picture elements.

2. The virtual input interface as recited in claim 1, wherein said input accepting means accepts input in the form of keystrokes.

3. The virtual input interface as recited in claim 1, wherein said input accepting means accepts input in the form of cursor position.

4. The virtual input interface as recited in claim 1, wherein said input accepting means accepts input in the form of system-defined actions.

5. The virtual input interface as recited in claim 1, wherein said input accepting means accepts input in the form of user-defined functions.

6. The virtual input interface as recited in claim 1, wherein said input accepting means accepts input in the form of menu selections.

7. The virtual input interface as recited in claim 1, wherein said at least one physical device can be removed from said system without affecting the operation of the remainder of said system.

8. The virtual input interface as recited in claim 1, wherein at least one additional physical device can be added to said system without affecting the operation of the remainder of said system.

9. A virtual output interface in a data processing system, said interface comprising:

a source of virtual input, said virtual input comprising one or more picture elements, each picture element comprising a plurality of device-independent data structures in a predetermined, standard data format, at least one of said data structures comprising a plurality of different data fields each containing information describing said picture element;

means for performing processing operations on said virtual input and for generating virtual output;

means for accepting said virtual output; and

means for converting said virtual output into at least one physical output suitable for use by at least one physical output device.

10. The virtual output interface as recited in claim 9, wherein said virtual input comprises a plurality of related picture elements and wherein said virtual output accepting means comprises a picture manager process for controlling said plurality of related picture elements.

226

11. The virtual output interface as recited in claim 10 and further comprising a display device, wherein said virtual output accepting means further comprises a window manager process for controlling the display of said plurality of related picture elements on said display device.

12. The virtual output interface as recited in claim 9, wherein said virtual output converting means comprises a virtual output manager process responsive to said one or more processed picture elements for coupling said one or more processed picture elements to said at least one physical output device.

13. The virtual output interface as recited in claim 9, wherein said at least one physical device can be removed from said system without affecting the operation of the remainder of said system.

14. The virtual output interface as recited in claim 9, wherein at least one additional physical device can be added to said system without affecting the operation of the remainder of said system.

15. In a data processing system, an interface between processes and data in said system and physical input and output devices coupled to said system, said interface comprising:

means responsive to one of said physical input devices for generating a picture, said picture comprising one or more picture elements, each picture element comprising a plurality of device-independent data structures in a predetermined, standard data format, at least one of said data structures comprising a plurality of different data fields each containing information describing said picture element;

means for performing processing operations on said one or more picture elements; and

means responsive to said one or more processed picture elements for coupling said one or more processed picture elements to one of said physical output devices.

16. The data processing system as recited in claim 15, wherein said one or more picture elements define a graphical object and at least one attribute thereof.

17. The data processing system as recited in claim 16, wherein one of said data fields describes the length of the associated picture element.

18. The data processing system as recited in claim 16, wherein one of said data fields identifies the particular type of the associated picture element.

19. The data processing system as recited in claim 16, wherein one of said data fields describes the position of the associated picture element relative to row and column coordinates on a picture of which said picture element forms a part.

20. The data processing system as recited in claim 16, wherein one of said data fields describes the size of the associated picture element.

21. The data processing system as recited in claim 16, wherein one of said data fields describes the color of the associated picture element.

22. The data processing system as recited in claim 15, wherein said means responsive to one of said physical input devices comprises a virtual input manager process.

23. The data processing system as recited in claim 15, wherein said means responsive to said one or more processed picture elements comprises a virtual output manager process.

* * * * *